



Office de la propriété  
intellectuelle  
du Canada

Un organisme  
d'Industrie Canada

Canadian  
Intellectual Property  
Office

An Agency of  
Industry Canada

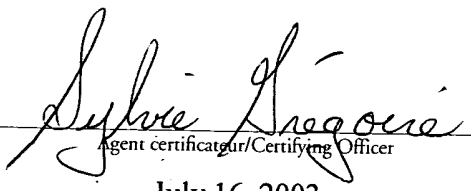
*Bureau canadien  
des brevets  
Certification*

La présente atteste que les documents  
ci-joints, dont la liste figure ci-dessous,  
sont des copies authentiques des docu-  
ments déposés au Bureau des brevets.

*Canadian Patent  
Office  
Certification*

This is to certify that the documents  
attached hereto and identified below are  
true copies of the documents on file in  
the Patent Office.

Specification and Drawings, as originally filed, with Application for Patent Serial No:  
**2,414,047**, on December 9, 2002, by **COREL CORPORATION**, assignee of Gordon  
Bowman and Peter Barrett, for "System and Method of Extending Scalable Vector Graphics  
Capabilities".

  
Agent certificateur/Certifying Officer  
July 16, 2003  
Date

Canada

(CIPO 68)  
04-09-02

OPIC  CIPO

**Abstract**

A system of manipulating a document object model of a web application. The system comprises a collection of designated elements, a collection of associated  
5 instructions for performing functions to elements in the document object model, the instructions associated with the designated elements, and an initialization function for directing the processing one or more designated elements in the document object model. Each designated element comprises a name following a predetermined naming convention, and attributes for describing features of the designated element.

## **System and Method for Extending Scalable Vector Graphics Capabilities**

### FIELD OF THE INVENTION

The invention relates to markup languages. In particular, the invention relates to a  
5 system and method for extending scalable vector graphics capabilities.

### BACKGROUND OF THE INVENTION

Scalable Vector Graphics (SVG) has the potential to become the platform (markup  
language) of choice for building robust, dynamic and interactive web applications.  
10 However, SVG lacks many features that are desired for building such web applications.  
Features that are missing in SVG include coordinate mapping, projection mapping, mouse  
tracking, zooming, panning, playing a sound, selection ability, flow control, moving  
objects, zoom/pan immunity, z-order, and constraints.

Since these features are missing from SVG, building robust web applications  
15 requires extensive scripting via ECMAScript. There are many problems with using script.  
One problem is the fact that most web designers do not have the programming skills  
required for scripting. One way of assisting designers and developers is to have pre-  
canned scripts for the most commonly required functionality. However, supporting the  
insertion of pre-canned scripts via an integrated development environment (IDE) is both  
20 complicated and limiting. For example, the Microsoft Visual Studio (TM) IDE can create  
auto-generated code for its Microsoft Foundation Class (MFC) (which abstract the  
programmer from the core Win32API's), making it easier and quicker to program  
Windows applications. However, limits must be imposed on the user. User-modification  
of the auto-generated code is discouraged, because it makes it difficult to regenerate code  
25 from the project file, or to automatically modify the pre-generated code as a result of new  
user-defined parameters to the abstractions.

Software exists that allows one to map input extensible markup language (XML)  
to output markup. However, there is no software that allows the user to map input XML  
data to variables in pre-canned script functions.

30 It is difficult to generate data-driven script via the extensible stylesheet language  
transformation (XSLT), the most commonly used markup language for transforming  
XML markup to a different form of markup.

Scripts are interpreted, and thus provide inherently slower performance than what can be achieved with natively implemented code. Scripts can only manipulate the document object model (DOM) via the DOM application programming interfaces (API's) that are exposed to the programmer, which may be abstractions on top of the real object model used by the viewer, which can only be access by native code. Scripts add to the amount of data needed to be transferred. This volume of data is especially a problem for wireless devices with low bandwidth. Finally, scripts are only as powerful as the DOM API's that the viewer supports. Currently, not all viewers support the entire spectrum of DOM API's. Thus, in order to ensure that the script will work on all viewers, one must write script that only uses the API's supported by all viewers.

The algorithm for determining the polynomial coefficients from a series of point-pairs is known as Singular Value Decomposition, which solves, in a least square sense, the overdetermined set of equations:

$$\begin{aligned}x_i' &= Ax_i + By_i + C \\ y_i' &= Dx_i + Ey_i + F\end{aligned}$$

given 3 or more coordinate pairs  $\{x_i', y_i' ; x_i, y_i\}$ .

The software EarthView, by Atlantis Scientific, has user interface (UI) for creating the point-pairs as well as a macro language for pulling the point-pairs in from a file, calculating the coefficients and transforming one coordinate space to another. EarthView can also convert between many different projection systems, using known algorithms. EarthView does not, however, support an XML markup language.

Figure 1 shows a typical web display environment 10 for displaying web pages and web applications. A web display environment 10 comprises a browser 11, a viewer 13, a script interpreter 14, and a DOM 15. The browser 11 is the host application, which understands and visually renders hypertext markup language (HTML) and/or extensible hypertext markup language (XHTML). Examples of browsers include Netscape (TM) and Internet Explorer (TM). The browser 11 includes a window which is displayed on the display apparatus, such as a monitor, of an end user computer system. The browser 11 typically employs a plug-in architecture, in which third party software (known as the plug-

in or viewer 13) can be associated with any file format that is not already natively supported by the browser 11 and is allowed to render that file within the host browser's 11 window. One type of file that the browser 11 may be asked to open is a Scalable Vector Graphic (SVG) file having a ".svg" extension. The browser 11 does not natively support the SVG markup language (which is an XML language) and so passes the SVG file to the SVG viewer 13, which has associated itself to the SVG file format, via the rules of the plug-in architecture of the browser 11.

The viewer 13 comprises software code for parsing the SVG markup, creating a DOM, rendering that DOM to the browser's window, listening for events and dispatching them to their assigned handler script functions, and interpreting/executing those script functions. An example of a viewer 13 is the Corel (TM) SVG Viewer. The viewer 13 uses the SVG file received from the browser 11 to create a DOM 15. The DOM is a hierarchical tree structure of objects in memory, representing the hierarchical XML markup in the XML text file. The DOM also contains methods (also known as functions or application programming interfaces (API's)) that allow it to be queried or modified. The viewer 13 may also have access to a script interpreter/engine, which can execute script code 14 created by a programmer for the purpose of making the document non-static (e.g., animation) and/or interactive with the user (e.g., the user can create events with the mouse or keyboard, which cause something to happen) via manipulation of the DOM.

## SUMMARY OF THE INVENTION

It is an object of the invention to provide a novel system and method of manipulating a document object model that obviates or mitigates at least one of the problems described above.

In an aspect of the present invention, there is provided a system of manipulating a document object model of a web application. The system comprises a collection of designated elements, a collection of associated instructions for performing functions to elements in the document object model, the instructions associated with the designated elements, and an initialization function for directing the processing one or more designated elements in the document object model. Each designated element comprises a

name following a predetermined naming convention, and attributes for describing features of the designated element.

In another aspect of the present invention, there is provided a method of controlling statement flow of a web application. The method comprises the steps of  
5 searching for a flow control element in a document object model of the web application, generating a function name associated with the flow control element, calling the generated function name and processing child elements of the flow control element.

In another aspect of the present invention, there is provided a method of coordinate mapping of a web application. The method comprises the steps of searching  
10 for a coordinate mapping element in a document object model of the web application, generating a function name associated with the coordinate mapping element, and calling the generated function name.

In another aspect of the present invention, there is provided a method of manipulating viewer behavior with respect to a web application. The method comprises  
15 the steps of searching for a viewer behavior element in a document object model of the web application, generating a function name associated with the viewer behavior element, and calling the generated function name.

In another aspect of the present invention, there is provided a method of selecting a group of element in a web application. The method comprises the steps of searching for  
20 a selection element in a document object model of the web application, generating a function name associated with the selection element, and calling the generated function name.

In another aspect of the present invention, there is provided a method of constraining manipulable attributes of an element in a web application. The method  
25 comprises the steps of searching for a constraint element in a document object model of the web application, generating a function name associated with the constraint element, and calling the generated function name.

In another aspect of the present invention, there is provided a method of applying passive behavior to an element of a web application. The method comprising the steps of  
30 searching for a designated attribute of the element in a document object model of the web

application, generating a function name associated with the designated attribute, and calling the generated function name.

In another aspect of the present invention, there is provided a method of manipulating a document object model. The method comprises the steps of searching for  
5 a designated control element in the document object model, and calling a function associated with the designated control element.

In another aspect of the present invention, there is provided a method of controlling features of a web application. The method comprises the steps of adding a behavior element as a child of a designated element, receiving an event which is equal to  
10 an event attribute setting in the behavior element, and calling a script associated with the behavior element.

In another aspect of the present invention, there is provided computer readable media storing the instructions and/or statements for use in the execution in a computer of a method of manipulating a document object model of a web application. The method  
15 comprises steps of searching for a designated element in a document object model, and calling a script associated with the designated element.

In another aspect of the present invention, there is provided electronic signals for use in the execution in a computer of a method of manipulating a document object model of a web application. The method comprising steps of searching for a designated element  
20 in a document object model, and calling a script associated with the designated element.

In another aspect of the present invention, there is provided a computer program product for use in the execution in a computer of a method for manipulating a document object model of a web application. The computer program product comprises a collection of functions for performing actions associated with designated elements, and an  
25 initialization function for directing the processing of one or more designated elements in a document object model.

## BRIEF DESCRIPTIONS OF THE DRAWINGS

Figure 1 shows a typical web display environment for displaying web pages and  
30 web applications.

Figure 2 shows an example of a scalable vector graphics capabilities extension system, in accordance with an embodiment of the present invention.

Figure 3 shows another example of a scalable vector graphics capabilities extension system, in accordance with an embodiment of the present invention.

5 Figure 4 is a flowchart of an example of a method of manipulating a document object model of a web application at load time, in accordance with an embodiment of the present invention.

Figure 5 is a flowchart of a method of a method manipulating a document object model of a web application in response to an event, in accordance with the an  
10 embodiment of the present invention.

Figure 6 is a flowchart of another example of an method of manipulating a document object model a web application, in accordance with an embodiment of the present invention.

## 15 DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figure 2 shows a system 20 for extending scalable vector graphics (SVG) capabilities, in accordance with an embodiment of the present invention. The SVG capability extension system 20 comprises an initialization file 21, one or more designated elements 29, and one or more associated scripts 28. Preferably, the one or more  
20 designated elements 29 map to the one or more associated scripts 28 on a one-to-one basis. The designated elements 29 comprise a name and attributes. The name of the designated elements follows a predetermined naming convention. For example, a prefix may be added to the generic name of the designated element 29. In one example of an embodiment of an SVG capability extension system 20, the prefix "dsvg:" is added to the  
25 generic name of the designated element 29. Among the attributes of the element are name & xmlns (identifying that the element belongs to the dsvg).

Figure 3 shows another example of an SVG capabilities extension system 30. The SVG extension system 30 comprises an initialization file 21, a collection of designated items 39 and a collection of associated scripts 38. The collection of designated items 39  
30 comprises one or more of the following: flow control elements 22, coordinate mapping elements 23, viewer behavior elements 24, a selection element 25, a constraint element



26, and a set of passive attributes 27. The collection of associated scripts 38 comprise flow control scripts 32, coordinate mapping scripts 33, viewer behavior scripts 34, a selection script 35, a constraint script 36, and a set of passive attributes script 37. Items 38, 39 may be added or removed from the SVG capability extension system 30. The  
 5 initialization file 21 contains instructions for traversing each node in a document object model (DOM) and for searching and calling functions associated with elements having names following a predetermined naming convention. The associated scripts 39 are matched with the designated elements 38 through the initialization function (or file) 21.

## 10 Flow Control Elements 22

Flow control is desired for building web applications. With scripting, programmers have conditional evaluative expressions, such as “if” and “switch” statements, and looping.

A flow control element 22 is used to control statement flow of the web  
 15 application. Flow control elements are used for conditional rendering of graphical elements or execution of behavior elements. Flow control elements 22 are inserted in a document object model (DOM) as parents of other DOM elements which are to be rendered or executed if the conditional flow statement of the parent flow control element is satisfied.

20 The attributes of a flow control element 22 may comprise one attribute having a complex expression representing the flow control statement. Preferably, multiple attributes are present, with each attribute representing one item in the expression representing the flow control statement. Having such a one-to-one mapping of multiple attributes to flow control statement items is advantageous for data mapping, where a web  
 25 application designer desires to define variables from one data type into extensible markup language (XML). The associated script 32 performs actions associated with the flow control element 22.

Flow control elements 22 include the ‘if’ element, the ‘switch’ element, the ‘case’ element, the ‘default’ element, and the ‘loop’ element. The ‘if’ element defines a simple  
 30 conditional statement which, if it evaluates to “true”, results in its child elements being executed or rendered. The ‘if’ element may be used in conjunction with a “% %” syntax

for referencing the value of another element's attribute. Such referencing of attributes and the "% %" syntax will be described further below. The following is the syntax of the 'if' flow control element 22:

```

5      <dsvg:if
      id="name"
      value1="string"
      op="(equal | notEqual | lessThan | greaterThan | lessThanOrEqual |
      greaterThanOrEqual)"
      value2="string"
10     />

```

The 'id' attribute is a standard XML attribute for assigning a unique "name" to an element. The 'id' attribute is optional. The 'value1' attribute specifies the first value to compare. The 'op' attribute specifies the operation to use in comparing two values. The 'value2' attribute specifies the second value to compare.

15 The 'switch' element defines a conditional statement, comparing one value to other values defined in a child 'case' element of the 'switch' element. The 'switch' element may be used in conjunction with a "% %" syntax for referencing the value of another element's attribute. Such referencing of attributes and the "% %" syntax will be described further below. The following is the syntax for the 'switch' element:

```

20     <dsvg:switch
      id="name"
      value="string"
      />

```

25 The 'id' attribute is a standard XML attribute for assigning a unique "name" to an element. The 'id' attribute is optional. The 'value' attribute specifies the value to compare against many others, which are defined in the child 'case' elements of the 'switch' element.

30 The 'case' element is a child of the 'switch' element. The 'case' element defines the value to compare to the 'switch' element's 'value' attribute. If comparison evaluates to "true", the child elements of the 'case' element are executed or rendered. The 'case' element may be used in conjunction with a "% %" syntax for referencing the value of

another element's attribute. Such referencing of attributes and the "% %" syntax will be described further below. The following is the syntax for the 'case' element:

```

5      <dsvg:case
        id="name"
        value="string"
        fallThrough="(true | false)"
      />

```

The 'id' attribute is a standard XML attribute for assigning a unique "name" to an element. The 'id' attribute is optional. The 'value' attribute specifies the value to compare against the 'switch' element's 'value' attribute. The 'fallThrough' attribute specifies whether execution should continue on to the child elements of the next <case> element or not.

The 'default' element is a child of the 'switch' element, which contains the action elements to be executed when all of the comparisons with the <case> elements have failed. The 'default' element may be used in conjunction with a "% %" syntax for referencing the value of another element's attribute. Such referencing of attributes and the "% %" syntax will be described further below. The following is the syntax for the 'default' element:

```

20     <dsvg:default
        id="name"
      />

```

The 'id' attribute is a standard XML attribute for assigning a unique "name" to an element. The 'id' attribute is optional by default.

The 'loop' element defines a loop or a repeated sequence of actions. The following is the syntax for the 'loop' element:

```

25     <dsvg:loop
        id="name"
        from="integer"
        to="integer"
        increment="integer"
30     value="integer"

```

```

elements="text"
elementsIDs="text"

```

```

/>

```

The 'id' attribute is a standard XML attribute for assigning a unique "name" to an  
 5 element. The 'id' attribute is optional. The 'from' attribute specifies the first value to  
 compare. The 'to' attribute specifies the last value to compare. The 'increment' attribute  
 specifies the amount to increment with each iteration. A default of "1" may be set. The  
 'value' attribute specifies the value for the current iteration. The 'elements' attribute is  
 the search string to compare against the 'id' attribute of every element in the DOM (or as  
 10 a child of a specified parent element. The 'elements' attribute can, and usually will,  
 contain the \* character to denote "any string". Anytime an element is found whose ID  
 matches this search string, the child behaviours will be executed. For example,  
 elementIDs="myCircle\*" would match elements with the ID's "myCircle1" and  
 "myCircleRed"; and elementIDs="\*Circle\*" would match elements with the ID's  
 15 "myCircle1" and "hisCircle2". The 'elementIDs' attribute is the search string to compare  
 against the 'id' attribute of every element in the DOM (or as a child of a specified parent  
 element. The 'elementIDs' attribute can, and usually will, contain the \* character to  
 denote "any string". Anytime an element is found whose ID matches this search string,  
 the child behaviours will be executed. For example, elementIDs="myCircle\*" would  
 20 match elements with the ID's "myCircle1" and "myCircleRed"; and  
 elementIDs="\*Circle\*" would match elements with the ID's "myCircle1" and  
 "hisCircle2".

### Coordinate Mapping Elements 23

25 Often a web application requires the ability to click and drag objects, perhaps for  
 the purposes of editing their positions. SVG does not currently have this capability.  
 Thus, script is required which tracks the mouse movements and creates a translation  
 transform on the object. Also, a user may desire to display coordinate information for  
 where the mouse cursor is located. Thus, the user first needs to know where the mouse  
 30 cursor is in the coordinate system of the SVG document, before the user can convert the

coordinate information to the coordinate system the user is using. Currently, only script can assist a user.

In order to display coordinate-based data, such as the location of cities on a background map, a user must convert those coordinates to the coordinate system of the SVG document. An example of such a conversion is a linear transformation (a scale and translation), such as a cartesian grid with parallel latitude and longitude lines. Another example of a transformation is a polynomial transformation, such as for a map with a latitude/longitude projection (i.e., curved lines of latitude, angled lines of longitude). Usually, calculations are required to determine transformations. SVG does not provide markup for creating and applying complex mathematical transformations.

Sometimes data visualized in a web application uses a projection system, such as a latitude/longitude or universal transverse mercator (UTM), which usually requires knowledge of how the projection system operates. To be able to map to the coordinate system of an SVG document usually requires knowledge of how to convert between such projections. SVG does not provide markup to specify a projection system and parameters, and automatically map any coordinate from one system to the other.

A coordinate mapping element 23 manipulates the coordinates of an object in the web application. Coordinate mapping elements 23 are used to display an object whose coordinates are in a system different from the DOM coordinate system. A coordinate mapping element 23 is inserted in a DOM as a child of another DOM elements.

The attributes of a coordinate mapping element 23 include point pair coordinates. The associate script 33 performs actions used to take third party XML data and position the data in a DOM.

Coordinate mapping elements 23 include the 'mousePosition' element, the 'mapCoords' element, the 'pointPair' element, and the 'mapProj' element. The 'mousePosition' element defines a container for holding the current mouse coordinates, relative to the document or to the parent element. The 'mousePosition' element is a child of an element other than the <svg> root element. The following is the syntax of the 'mousePosition' element:

```
30      <dsvg:mousePosition
          id="name"
```

```

x="<coordinate>"
y="<coordinate>"
absolute="(true | false)"

```

```

/>

```

- 5 The 'id' attribute is a standard XML attribute for assigning a unique "name" to an element. The 'id' attribute is optional. The 'x' attribute indicates the x-coordinate of the mouse cursor, in the SVG document's coordinate space. The 'x' attribute is a storage attribute, intended to be referenced. The 'y' attribute indicates the y-coordinate of the mouse cursor, in the SVG document's coordinate space. The 'y' attribute has no is a storage attribute, intended to be referenced. Referencing attributes will be described further below. The 'absolute' attribute specifies whether the mouse coordinates are relative to the document (true) or relative to the parent element (false).

- The 'mapCoords' element defines an object used for mapping coordinates in one space to another space, via a polynomial transformation, whose coefficients are determined by the coordinates of the point-pairs given in the child 'pointPair' elements. The following is the syntax for the 'mapCoords' element:

```

<dsvg:mapCoords
  id="name"
  order="integer"
  x="<coordinate>"
  y="<coordinate>"
  u="<coordinate>"
  v="<coordinate>"
  unitsXY="(pixels | latlong)"
  unitsUV="(pixels | latlong)"
  inputID="name"

```

```

/>

```

- The 'id' attribute is a standard XML attribute for assigning a unique "name" to an element. The 'id' attribute is optional. The 'order' attribute specifies the order of the polynomial transformation. The default may be set to "1", which only requires 2 point-pairs, resulting in an affine (linear) transformation. The 'x' attribute indicates the x-

coordinate of the first coordinate system. Updating the 'x' attribute automatically updates the 'u' attribute. The 'y' attribute indicates the y-coordinate of the first coordinate system. Updating the 'y' attribute automatically updates the 'v' attribute. The 'u' attribute indicates the x-coordinate of the second coordinate system. Updating the 'u' attribute automatically updates the 'x' attribute. The 'v' attribute indicates the y-coordinate of the second coordinate system. Updating the 'v' attribute automatically updates the 'y' attribute. The 'unitsXY' attribute specifies the units of the x-y coordinates. The 'unitsUV' attribute specifies the units of the u-v coordinates. The 'inputID' attribute specifies the ID of the element that will feed its coordinates into the 'mapCoords' element whenever an update occurs. For example, the inputID may be the ID of a 'mousePosition' element.

The 'pointPair' element, which is a child of the 'mapCoords' element, defines the x-y coordinates for the same location in two different coordinate spaces. The point-pairs are used to calculate the polynomial transformation coefficients. The following is the syntax for the 'pointPair' element:

```
<dsvg:pointPair
    id="name"
    x="<coordinate>"
    y="<coordinate>"
    u="<coordinate>"
    v="<coordinate>"
/>
```

The 'id' attribute is a standard XML attribute for assigning a unique "name" to an element. The 'id' attribute is optional. The 'x' attribute indicates the x-coordinate of the first coordinate system. Updating the 'x' attribute automatically updates the 'u' attribute. The 'y' attribute indicates the y-coordinate of the first coordinate system. Updating the 'y' attribute automatically updates the 'v' attribute. The 'u' attribute indicates the x-coordinate of the second coordinate system. Updating the 'u' attribute automatically updates the 'x' attribute. The 'v' attribute indicates the y-coordinate of the second coordinate system. Updating the 'v' attribute automatically updates the 'y' attribute.

The 'mapProj' element defines an object used for mapping coordinates in one projection system (e.g., latitude/longitude) to another (e.g., UTM). The following is the syntax for the 'mapProj' element:

```

5      <dsvg:mapProj
      id="name"
      projXY="(LatLong | UTM)"
      projUV="(LatLong | UTM)"
      ellipsoid="(Airy | Australian National | Bessel 1841 | Bessel 1841
      (Nambia) | Clarke 1866 | Clarke 1880 | Everest | Fischer 1960 (Mercury) |
10     Fischer 1968 | GRS 1967 | GRS 1967 | GRS 1980 | Helmert 1906 | Hough,
      International | Krassovsky | Modified Airy | Modified Everest | Modified
      Fischer 1960 | South American 1969 | WGS 60 | WGS 66 | WGS-72 |
      WGS-84)"
      zone="string"
15     x="<coordinate>"
      y="<coordinate>"
      u="<coordinate>"
      v="<coordinate>"
      inputID="name"
20     />

```

The 'id' attribute is a standard XML attribute for assigning a unique "name" to an element. The 'id' attribute is optional. The 'projXY' attribute specifies the projection system of the x-y coordinate space. The 'projUV' attribute specifies the projection system of the u-v coordinate space. The 'ellipsoid' attribute specifies the ellipsoid of the UTM projection system. The 'zone' attribute specifies the zone of the UTM projection system. The 'x' attribute indicates the x-coordinate of the first coordinate system. Updating the 'x' attribute automatically updates the 'u' attribute. The 'y' attribute indicates the y-coordinate of the first coordinate system. Updating the 'y' attribute automatically updates the 'v' attribute. The 'u' attribute indicates the x-coordinate of the second coordinate system. Updating the 'u' attribute automatically updates the 'x' attribute. The 'v' attribute indicates the y-coordinate of the second coordinate system.



Updating the 'v' attribute automatically updates the 'y' attribute. The 'inputID' attribute specifies the ID of the element that will feed its coordinates into the 'mapProj' element whenever an update occurs. For example, the 'inputID' attribute may be the ID of a 'mapCoords' element.

5

#### Viewer Behavior Elements 24

SVG viewers allow a user to zoom in and out, and to pan, using a built-in UI, or by accessing the SVG DOM via script. There currently does not exist an SVG DOM API for playing a sound. The only way to play a sound currently is via an Adobe extension in the Adobe SVG Viewer.

10

A viewer behavior element 24 is used to manipulate viewer behavior with respect to the web application. An example of a viewer is an SVG plugin for a web browser. In particular, the viewer behavior elements 24 assist a designer to zoom and pan the current document or a document fragment.

15

The attributes of a viewer behavior element 24 include an event attribute that triggers the viewer behavior element. The associated script 34 performs actions used to zoom and pan a document in a web application. Other actions may be performed by the associated implementation code 34.

Viewer behavior elements 24 include the 'zoom' element, the 'pan' element, and the 'playSound' element. The 'zoom' element scales the document by a desired factor. The following is the syntax for the 'zoom' element:

20

```
<dsvg:zoom
    id="name"
    event="string"
    scale="integer"
    absolute="(true | false)"
    cx="<coordinate>"
    cy="<coordinate>"
    (target="<xpath>" | frameID="string" | objectID="string" |
    docID="string")
/>
```

25

30

The 'id' attribute is a standard XML attribute for assigning a unique "name" to an element. The 'id' attribute is optional. The 'event' attribute specifies the event that will trigger this action. The 'scale' attribute specifies the scale factor by which to zoom in or out. A factor greater than "1" results in zooming in. A factor less than "1" results in zooming out. The 'absolute' attribute specifies whether to set the document's scale factor to 'scale' (true) or to its current scale factor multiplied by 'scale' (false). The 'cx' attribute specifies the x-coordinate of the location in the document that will stay preserved after the zoom, with respect to the browser window. The 'cy' attribute specifies the y-coordinate of the location in the document that will stay preserved after the zoom, with respect to the browser window. The 'target' attribute specifies the target document of which to set the scale factor. A default may be set to the current document.

The 'pan' element translates the document by a desired amount. The following is the syntax for the 'pan' element:

```

15      <dsvg:pan
          id="name"
          event="<string>"
          x="integer"
          y="integer"
          absolute="(true | false)"
20      />

```

The 'id' attribute is a standard XML attribute for assigning a unique *name* to an element. The 'id' attribute is optional. The 'event' attribute specifies the event that will trigger this action. The 'x' attribute specifies the amount to translate horizontally. The 'y' attribute specifies the amount to translate vertically. The 'absolute' attribute specifies whether to set the document's translation to ('cx', 'cy') (true) or to its current translation plus ('cx', 'cy').

The 'playSound' element plays an audio file. The following is the syntax for the 'playSound' element:

```

30      <dsvg:playSound
          id="name"
          event="string"

```

```
xlink:href="<uri>"
```

```
/>
```

The 'id' attribute is a standard XML attribute for assigning a unique *name* to an element. The 'id' attribute is optional. The 'event' attribute specifies the event that will trigger this action. The 'xlink:href' attribute specifies the audio file to play.

### The Selection Element 25

Often in an application, a user desires to select an object or group of objects, via clicking or dragging a window around them, or even displaying a marquee around the selected object(s).

The selection element 25 is used to select a group of elements in a web application. The 'selection' element 25 defines a group of elements. Whenever an element, whose 'selectionGroup' attribute is equal to the 'id' of the selected group, is clicked on, the appearance of the other element will change according to the skin of the selection element 25 and this element's child action elements will be executed. The following is the syntax for the selection element 25:

```
<dsvg:selection element
    id="name"
    xlink:href="<uri>"
    multiSelect="true | false"
/>
```

The 'id' attribute is a standard XML attribute for assigning a unique "name" to an element. The 'id' attribute is optional. The 'xlink:href' attribute is a reference to the skin's parent element, stored either internally in the <defs> block, or in an external file. The 'multiSelect' attribute specifies whether to allow multiple elements to be selected (true) or not (false).

### The Constraint Element 26

One feature missing from SVG markup is the concept of constraints. A designer may desire zoom-dependent visibility of labels on a map. Another desire may be to have

the coordinates or dimensions of one element to be dependent on the coordinates or dimensions of another element.

The constraint element 26 is used to constrain manipulable attributes of an element in a web application. The 'constraint' element defines the rules for constraining the attributes of an element. Typically, a constraint element 26 is a child of the element whose attributes it constrains. The constraint element 26 is triggered by a transformation or zoom or a mutation event. The constraint element 26 could also be included as a child behaviour of an element (or within an action block) to be executed in sequence. The following is the syntax of the constraint element 26:

```

10      <dsvg:constraint
        id="name"
        {target="xpath" | frameID="integer" objectID="integer" docID="integer"
        elementID="integer"}
        attributeName="string"
15      propertyName="string"
        value="string"
        scaleImmunity="{true | false }"
        zoomImmunity="{true | false }"
        preserveAspectRatio="{vertical | horizontal | none }"
20      hAlign="{left | middle | right }"
        vAlign="{top | middle | bottom }"
        width="integer"
        height="integer"
        left="integer"
25      right="integer"
        top="integer"
        bottom="integer"
        />

```

The 'id' attribute is a standard XML attribute for assigning a unique name to an element.

30 The 'id' attribute is optional. The 'target' attribute is the target element whose attribute or style property is to be constrained. The 'attributeName' attribute is the name of the

attribute to be constrained. The 'propertyName' attribute is the name of the style property (e.g., 'stroke-width') to be constrained. The 'value' attribute is the value that the attribute defined by 'attributeName', or the style property defined by 'propertyName', is to be given. The 'scaleImmunity' attribute specifies that the attribute or style property of the target element should be immune to scaling (via a transform). The 'zoomImmunity' attribute specifies that the attribute or style property of the target element should be immune to zooming (via a transform). The 'preserveAspectRatio' attribute specifies which dimension is to be preserved, thus altering the other dimension so as to preserve the original aspect ratio, which was altered presumably by a transformation. The 'hAlign' attribute is the part of the target element, along the x-axis, that is to have its position preserved after executing the constraint. For example, preserveAspectRatio="vertical" might cause the target element to be scaled along the x-axis, causing its horizontal position to change. Specifying hAlign="right" would cause the right edge of the target element to stay where it was before the transformation that prompted the constraint to be called. The 'vAlign' attribute is the part of the target element, along the y-axis, that is to have its position preserved after executing the constraint. For example, preserveAspectRatio="horizontal" might cause the target element to be scaled along the y-axis, causing its horizontal position to change. Specifying vAlign="top" would cause the top edge of the target element to stay where it was before the transformation that prompted the constraint to be called. The 'width' attribute is the width that the target element is to be. For example, if the target element is a 'g' tag (a group) containing many elements, the total width of all those elements put together is not obvious to a designer. Thus it is not easy for a designer to determine the scale factor along the x-axis required to achieve that width. Setting the width using the 'constraint' element performs this task for the designer. The 'height' attribute is the height that the target element is to be. For example, if the target element is a 'g' tag (a group) containing many elements, the total height of all those elements put together is not obvious to a designer. Thus, it is not easy for the designer to determine the scale factor along the y-axis required to achieve that height. Setting the height using the 'constraint' element performs this task for the designer. The 'left' attribute is the x-coordinate at which the left edge of the target element is to be. For example, if the target element is a 'g' tag (a group) containing many

elements, the left edge of all those elements put together is not obvious to a designer. Thus, it is not easy for a designer to determine the translation along the x-axis required. Setting the coordinates of the left edge using the 'constraint' element performs this task for the designer. If the 'right' attribute is also specified, both edges will be set, which will likely also require a scale transformation. The 'right' attribute is the x-coordinate at which the right edge of the target element is to be. For example, if the target element is a 'g' tag (a group) containing many elements, the right edge of all those elements put together is not obvious to a designer. Thus, it is not easy for the designer to determine the translation along the x-axis required. Setting the coordinates of the right edge using the 'constraint' element performs this task for the designer. If the 'left' attribute is also specified, both edges will be set, which will likely also require a scale transformation. The 'top' attribute is the y-coordinate at which the top edge of the target element is to be. For example, if the target element is a 'g' tag (a group) containing many elements, the top edge of all those elements put together is not obvious to a designer. Thus, it is not easy for a designer to determine the translation along the x-axis required. Setting the coordinates of the top edge using the 'constraint' element performs this task for the designer. If the 'bottom' attribute is also specified, both edges will be set, which will likely also require a scale transformation. The 'bottom' attribute is the y-coordinate at which the bottom edge of the target element is to be. For example, if the target element is a 'g' tag (a group) containing many elements, bottom edge of all those elements put together is not obvious to a designer. Thus, it is not easy for a designer to determine the translation along the x-axis required. Setting the coordinates of its bottom edge using the 'constraint' element performs this task for the designer. If the 'top' attribute is also specified, both edges will be set, which will likely also require a scale transformation.

### Passive Attributes 27

A common requirement in a web application is to be able to zoom and pan on the content without the UI controls zooming and panning. This capability does not exist in SVG markup. Scripting is required to detect the SVGScale, SVGScroll and SVGResize events, and create a transformation on all elements that should be immune to zooming and panning, which will counteract the zoom or pan.

A passive attribute 27 is applied to one or more DOM elements for applying passive behavior to objects in a web application. By adding a passive attributes 27 to an element, other elements can then reference the passive attribute 27. Referencing attributes will be discussed below.

5 For example, if there is a legend placed in the bottom corner of the screen on a map, the 'g' (group) element that contains the legend subtree could be given the passive attributes zoom="false" and pan="false". Thus, if the map were then zoomed or panned, the legend would continue to stay at the bottom corner of the screen.

The following attributes can be applied to an element in a DOM:

10

drag="(true | false)" Specifies whether the element is movable (true) or not (false) by clicking and dragging it with the mouse.

15

pan="(true | false)" Specifies whether the element is immune to panning (false) or not (true).

zoom="(true | false)" Specifies whether the element is immune to zooming (false) or not (true).

20

selected="(true | false)" Specifies whether the element has been selected or not. Clicking on any object which has the 'selectionGroup' attribute, which causes it to become selected and might cause other objects to become unselected, will cause the 'selected' attribute to be updated appropriately.

25

selectionGroup="string" Specifies the 'id' attribute of the 'selection' element that this element is associated with.

30

Other items may be added to the collection of designated items. For example, behavior elements and user interface control elements may be added along with their associated scripts.

In the method (40) described above, the function was dynamically generated, i.e.,  
 5 a string was created, having the same prefix as the designated element (without the colon) and the same name as the designated element (except with the first letter capitalized) and with the designated element's object and the trigger event object passed in as two parameters. The associated script 38 or set of instructions for the operations of the generated function is stored in a predetermined format either in the document text file or  
 10 in a separate text file on a file system or webserver, and is loaded into memory by the viewer at load time. Alternatively, the initialization function may search for elements that begin with the "dsvg:" prefix and, using an 'if' or 'switch' statement, determine the appropriate predetermined function to call, which again are expected to have been already loaded in memory by the viewer.

15 It is advantageous, though, for the function names to be generated dynamically, so that the main script file containing the initialization function 21 does not need to be updated whenever a new type of designated element 39 has been created and is available for use.

As well, while the functions 38 that handle each type of designated element 38  
 20 could be stored all in one file, it is advantageous to store them in separate files and reference them in the document only if their corresponding designated element 38 is being used, so that only the code that is required is actually transmitted.

In order for designated elements 38 to execute desired actions, behavior elements may be inserted as children of the designated elements 38 (the observer elements). The  
 25 behavior element will be executed sequentially for each behavior element whose 'event' attribute's value matches the observer element's event (e.g., onmouseover, onclick, etc.). If the 'event' attribute is not provided, the behavior will default to be run on the 'onclick' event. In the example below, clicking on the 'buttonZoomIn' button will cause the processing of the child elements of the button, and the zoom behavior will be executed,  
 30 scaling the document 2x, while clicking on the 'buttonZoomOut' button will scale the document 0.5x.



```

<dsvg:button id="buttonZoomIn" x="10" y="10" label="Zoom In"
xlink:href="#skinZoomInButton">
    <zoom scale="2"/>
</dsvg:button>

```

5

```

<dsvg:button id="buttonZoomOut" x="10" y="40" label="Zoom Out"
xlink:href="#skinZoomOutButton">
    <zoom scale="0.5"/>
</dsvg:button>

```

10

Alternatively, the behavior elements may be grouped as children of an <action> element (or behavior element), which can be hooked up to the observer element using a <listener> element. For example:

```

<dsvg:button id="buttonZoomIn" x="10" y="10" label="Zoom In"
xlink:href="#skinZoomInButton"/>
<dsvg:button id="buttonZoomOut" x="10" y="40" label="Zoom Out"
xlink:href="#skinZoomOutButton"/>

```

15

```

<dsvg:action id="zoomIn">
    <zoom scale="2"/>
</dsvg:action>

```

20

```

<dsvg:action id="zoomOut">
    <zoom scale="0.5"/>
</dsvg:action>

```

25

```

<dsvg:listener event="onclick" observerElementID="buttonZoomIn"
handlerID="zoomIn"/>
<dsvg:listener event="onclick" observerElementID="buttonZoomOut"
handlerID="zoomOut"/>

```

30

Thus, during the document load, the onclick event of the buttonZoomOut element is associated with the zoomOut action, via the listener that identifies observerElementID="buttonZoomOut". When this button is clicked, the children of the zoomOut action will be processed, scaling the document x0.5.

5        Figure 5 shows an example of a method of manipulating a DOM of a web application in response to an event (50) in accordance with the SVG capabilities extension system 20, 30. The SVG capabilities extension system 20, 30 is built on top of an event-driven architecture, such as SVG, and XML. Once an event occurs on an SVG element (i.e., the observer element), the method (50) begins with passing the event object  
10        to a handler function (51). The handler function determines if the first child element of the SVG element associated with the object is a designated element (52). If a designated element is found (53), then the handler function determines if the event attribute 24 of the designated element is equal to the event that has occurred (54). If the event attribute 24 of the designated element is equal to the event which triggered this method (50), then the  
15        name of the function associated with the designated element is automatically generated (55) (in accordance with a predetermined function naming convention) and called (56). Preferably, the predetermined function naming convention is similar to the predetermined element naming convention. If a designated element is not found (53), or if the event attribute 24 of the designated element does not match the trigger event (54), or after a  
20        generated function is called (56), the event handler determines if there are more child elements of the observer element to search (57). If there are more child elements of the observer element (57), the event handler determines if the next child is a designated element (58). Steps (53) to (58) are repeated until all child elements of the observer element are searched. Once there are no more child elements to search (57), then the  
25        handler function is done (59).

The initialization file 21 may also search for designated attributes in SVG elements. Scripts 38 may be created and associated with the 'dsvg' attribute in the same manner as with UI control elements. Script functions 38 for 'dsvg' attributes only operate on the object associated with the existing element to which a 'dsvg' attribute is added. A  
30        designer may add the 'dsvg' attribute in an SVG file, or any other XML file to be parsed by the viewer 13.

Figure 6 shows another example of a method of manipulating a DOM of a web application (60), in accordance with the SVG capability extension system 20, 30. After a user (or designer) marks up an SVG file using the markup syntax of the SVG capabilities extension system 20, 30 and the SVG file is loaded into a viewer 13, the viewer 13 creates an “onload” event which is received by an <svg> element. The method (60) begins with the initialization function 21. A dsvgInit() initialization function 21 is called (61) by the viewer’s script interpreter, which traverses the nodes of the DOM of the SVG file. The initialization function 21 determines if the first DOM element is a designated element 29, 39 (62). If a designated element 39 is found (63) and the ‘event’ attribute of the designated element 28 is set to “onload” (64), then the name of the function 38 associated with the designated element 29, 39 is automatically generated (65) (in accordance with a predetermined function naming convention) and called (66). Preferably, the predetermined function naming convention is similar to the predetermined element naming convention. If a designated element 29, 39 is not found (63), the initialization function 21 determines if the regular SVG element contains any designated attributes 27 (67) (which begin with the “dsvg:” prefix). If any designated attributes 27 are found (67) (e.g., dsvg:toolTip="#skinTooltip\_traditional”), then the names of the functions 37 associated with the designated attributes are automatically generated (68) (again, in accordance with a predetermined function naming convention) and called (69).

If a designated attribute 27 is not found (67), then the initialization file 21 determines if the regular SVG element has any child elements (70). If the regular SVG element has a child element (70) and the child element is a designated element 29, 39 (71), then the initialization file 21 determines the value of the designated element’s ‘event’ attribute (i.e., the event that will trigger the execution of the designated element’s associated function) and adds that event listener to the parent SVG element (72) (via the addEventListener() DOM API). If the child element is not a designated element 29, 39 (71), then the initialization file 21 determines if there are any other children of the regular SVG element (73). If there are more children (73), then the initialization file searches the next child of the regular SVG element (74). Steps (71) to (74) repeat until there are no more children of the regular SVG element.

If there are no more children of the regular SVG element (73), or after a generated function is called (76, 79), or if the event attribute of a designated element is not equal to "onload" (64), or there are no more child elements in a regular SVG element to search (70), the initialization file 21 determines if there are more elements in the DOM to search (75). If there are more elements in the DOM (75), the initialization file determines if the next sibling element is a designated element (76). Steps (73) to (76) are repeated until all elements in the DOM are searched. Once there are no more elements in the DOM to search (75), then the initialization function 21 is done and the viewer 13 waits for an event to occur (77).

- 10 Once an event occurs on an SVG element (i.e., the observer element), that event object is passed to a handler function with which it has been associated (78). The handler function determines if any child of the observer element is a designated element 29, 39 (79). The event handler function calls the appropriate script 28, 38 (as described in Figure 6) for any child of the observer element that is a designated element 29, 39 (80).
- 15 Once all children of the observer element are processed (80), then the event handler function is done and the viewer waits for another event to occur (77).

### Referencing Attributes

- To create an application, a designer often desires to reference the current value of another element's attributes. The system 20, 30 allows for the following syntax to perform this reference:

`%frameID.objectID.docID.elementID.childElementID@attributeName%`

- 25 If not specified, the 'frameID', 'objectID' and 'docID' are assumed to be the current frame, object and document. In the following example, checking and unchecking the checkbox will disable and enable the textbox.

30 `<dsvg:button id="myCheckBox" x="10" y="10"  
toggle="true" label="Disable slider"  
xlink:href="#skinCheckbox">`

```

        <dsvg:setAttribute elementID="myTextBox"
name="disabled" value="%myCheckBox.checked%"/>
</dsvg:button>

```

```

5      <dsvg:textBox id="myTextBox" />

```

The system 20, 30 also allows for parenthesis and mathematical operators within the % % expression. For example, in the markup:

```

10    <dsvg:button id="button" x="50" y="50" label="foo"
toggle="true" group="pickPanGroup"
xlink:href="skinButton_Windows.svg#skinButton"
selected="true" />

    <dsvg:button id="button_2" x="200" y="200" label="foo"
15    toggle="true" group="pickPanGroup"
xlink:href="skinButton_Windows.svg#skinButton"
selected="true" />

```

an attribute "foo %button\_(button@x - 48)@x + 14 \* button@y% bar%button@x%" is

20 parsed as follows:

```

'foo %button_(button@x - 48)@x + 14 * button@y% bar%button@x/2%'
-> 'foo %button_(50 - 48)@x + 14 * button@y% bar%button@x/2%'
-> 'foo %button_(2)@x + 14 * button@y% bar%button@x/2%'
25 -> 'foo %button_2@x + 14 * button@y% bar%button@x/2%'
-> 'foo %200 + 14 * 50% bar%button@x/2%'
-> 'foo %200 + 700% bar%button@x/2%'
-> 'foo 900 bar%button@x/2%'
-> 'foo 900 bar25'

```

30 There are many advantages to the SVG capabilities extension system 20, 30. The SVG capabilities extension system 20, 30 assists web designers with no programming

skills to create dynamic, interactive web applications. It also aids experienced programmers to create dynamic, interactive web applications much more easily and rapidly. Because the SVG capabilities extension system 20, 30 involves an XML markup language (as opposed to just script functions), the attributes and data and even the elements themselves can be made to be data-driven at run-time, using (at design-time) existing or new software that allows one to visually map input XML markup to output XML markup, resulting in an extensible stylesheet language transformation (XSLT) code (or any other language useful for XML transformations) which will actually modify the designated elements 29, 39 based on the input XML data/markup.

The SVG capabilities extension system 20, 30 can also be natively-implemented, accessing the exposed DOM API's in the same manner as the script implementation. A native implementation could be much faster because unlike script, which gets interpreted at run-time, native code (e.g. C++ or C) gets interpreted at compile time and gets optimized by the compiler. The natively-implemented SVG capabilities extension system 20, 30 could also access any unexposed, lower-level object model API's directly, rather than the exposed higher-level DOM API's, which could further improve performance. If natively implemented, the amount of data needed to be transferred may be greatly reduced, since there is no script that needs to be transmitted, which is especially beneficial for wireless devices with low bandwidth and small memory. Using a markup language for the designated elements 29, 39 is also beneficial because it allows for the possibility of further reducing the file size by creating a binary version of the markup language that employs opcodes—predetermined arrangements of bits (1's and 0's) that correspond to particular element names and attributes. Unlike textual markup, which must be parsed (compared to predetermine strings/text to establish the meaning of the text) in order to create the DOM, binary opcodes can be compared to identical binary opcodes, which is much faster than string comparisons, in order to build the DOM much faster.

The SVG capabilities extension system 20, 30 according to the present invention may be implemented by any hardware, software or a combination of hardware and software having the above described functions. The software code, either in its entirety or a part thereof, may be stored in a computer readable memory. Further, a computer data

signal representing the software code which may be embedded in a carrier wave may be transmitted via a communication network. Such a computer readable memory and a computer data signal are also within the scope of the present invention, as well as the hardware, software and the combination thereof.

- 5           While particular embodiments of the present invention have been shown and described, changes and modifications may be made to such embodiments without departing from the true scope of the invention.

## WHAT IS CLAIMED IS:

1. A system of manipulating a document object model of a web application, the system comprising:

- 5           a collection of designated elements, each designated element comprising:
- a name following a predetermined naming convention; and
- attributes for describing features of the designated element;
- a collection of associated instructions for performing functions to elements in the document object model, the instructions associated with the designated elements; and
- 10           an initialization function for directing the processing one or more designated elements in the document object model.

2. The system as claimed in claim 1, wherein the predetermined naming convention comprises having a constant prefix to the name of the element.

15

3. The system as claimed in claim 1, wherein the designated element is associated with an extensible markup language element.

4. The system as claimed in claim 1, further comprising:

- 20           a collection of designated attributes applied to one or more of the document object model elements for applying passive behavior to objects in the web application; and
- a collection of associated instructions for performing functions associated with the designated attributes.

25   5. The system as claimed in claim 4, wherein the collection of designated attributes comprises one or more of:



a 'drag' attribute for specifying whether the element is movable by clicking and dragging it with the mouse;

a 'pan' attribute for specifying whether the element is immune to panning;

a 'zoom' attribute for specifying whether the element is immune to zooming;

5 a 'selected' attribute for specifying whether the element has been selected; and

a 'selectionGroup' attribute for specifying an 'id' attribute of a <selection> element that this element is associated with.

6. The system as claimed in claim 1, wherein the collection of designated elements  
10 comprises one or more of:

flow control elements for controlling statement flow of the web application;

coordinate mapping elements for manipulating coordinates of objects in the web application;

behavior elements for manipulating viewer behavior with respect to the web  
15 application;

a <dsvg:selection> element for selecting a group of elements in the web application; and

a <dsvg:constraint> element for constraining manipulable attributes of an element in a web application.

20

7. The system as claimed in claim 6, wherein the flow control elements comprise one or more of:

a <dsvg:if> element for defining a simple conditional statement which, if it evaluates to true, results in its child elements being executed or rendered;

25 a <dsvg:switch> element for defining a conditional statement, and for comparing one value to other values defined in child <case> elements;

a <dsvg:case> element for defining the value to compare to a 'value' attribute of the <dsvg:switch> element;

a <dsvg:default> element for containing action elements to be executed; and

a <dsvg:loop> element for defining a repeated sequence of actions.

5

8. The system as claimed in claim 7, wherein the <dsvg:if> flow control element comprises:

an 'id' attribute for assigning a unique name to an element;

a 'value1' attribute for specifying a first value to compare;

10 an 'op' attribute for specifying an operation to use in comparing two values; and

a 'value2' attribute for specifying a second value to compare.

9. The system as claimed in claim 7, wherein the <dsvg:switch> flow control element comprises:

15 an 'id' attribute for assigning a unique name to an element; and

a 'value' attribute for specifies a value to compare against another value, the other value defined in a child <case> element.

10. The system as claimed in claim 7, wherein the <dsvg:case> flow control element  
20 comprises:

an 'id' attribute for assigning a unique name to an element;

a 'value' attribute for specifying a value to compare against a 'value attribute of a <switch> element; and

a 'fallThrough' attribute for specifies whether execution should continue on to the  
25 child elements of a next <item> element.

11. The system as claimed in claim 7, wherein the `<dsvg:case>` flow control element is a child of the `<dsvg:switch>` flow control element.
12. The system as claimed in claim 7, wherein the `<dsvg:default>` flow control element  
5 comprises an 'id' attribute for assigning a unique name to an element.
13. The system as claimed in claim 7, wherein the `<dsvg:default>` flow control element is a child of the `<dsvg:switch>` flow control element.
- 10 14. The system as claimed in claim 7, wherein the `<dsvg:loop>` flow control element comprises:
- an 'id' attribute for assigning a unique name to an element;
  - a 'from' attribute for specifying a first value to compare;
  - a 'to' attribute for specifying a last value to compare;
  - 15 an 'increment' attribute for specifying an amount to increment with each iteration;
  - a 'value' attribute for specifying the value for a current iteration;
  - an 'elements' attribute for comparing against an 'id' attribute of elements in the document object model; and
  - an 'elementIDs' attribute for comparing against an 'id' attribute of elements in the  
20 document object model.
15. The system as claimed in claim 6, wherein the coordinate mapping elements comprise one or more of:
- a `<dsvg:mousePosition>` element for defining a container for holding current  
25 mouse coordinates;

a <dsvg:mapCoords> element for defining an object used for mapping coordinates in one space to another space, via a polynomial transformation, whose coefficients are determined by the coordinates of point-pairs;

5 a <dsvg:pointPair> element for defining x-y coordinates for a same location in two different coordinate spaces; and

a <dsvg:mapProj> element for defining an object used for mapping coordinates in one projection system to another.

10 16. The system as claimed in claim 15, wherein the <dsvg:mousePosition> coordinate mapping element comprises:

an 'id' attribute for assigning a unique name to an element;

an 'x' attribute for indicating an x-coordinate of a mouse cursor, in a scalable vector graphics document coordinate space;

15 a 'y' attribute for indicates a y-coordinate of the mouse cursor, in the scalable vector graphics document coordinate space; and

an 'absolute' attribute for specifying whether the mouse coordinates are relative to the document or relative to a parent element.

20 17. The system as claimed in claim 15, wherein the <dsvg:mapCoords> coordinate mapping element comprises:

an 'id' attribute for assigning a unique name to an element;

an 'order' attribute for specifying an order of a polynomial transformation;

an 'x' attribute for indicating an x-coordinate of a first coordinate system;

a 'y' attribute for indicating a y-coordinate of the first coordinate system;

25 a 'u' attribute for indicating an x-coordinate of a second coordinate system;

a 'v' attribute for indicating a y-coordinate of the second coordinate system;

a 'unitsXY' attribute for specifying units of the x-y coordinates;

a 'unitsUV' attribute for specifying units of the u-v coordinates; and  
 an 'inputID' attribute for specifying the identification of an element that will feed  
 its coordinates into the <mapCoords> element.

5 18. The system as claimed in claim 15, wherein the <dsvg:pointPair> coordinate  
 mapping element comprises:

an 'id' attribute for assigning a unique name to an element;  
 an 'x' attribute for indicating an x-coordinate of a first coordinate system;  
 a 'y' attribute for indicating a y-coordinate of the first coordinate system;  
 10 a 'u' attribute for indicating an x-coordinate of a second coordinate system; and  
 a 'v' attribute for indicating a y-coordinate of the second coordinate system.

19. The system as claimed in claim 15, wherein the <dsvg:mapProj> coordinate mapping  
 element comprises:

15 an 'id' attribute for assigning a unique name to an element;  
 a 'projXY' attribute for specifying a projection system of an x-y coordinate space;  
 a 'projUV' attribute for specifying a projection system of a u-v coordinate space;  
 an 'ellipsoid' attribute for specifying an ellipsoid of a UTM projection system;  
 a 'zone' attribute for specifying a zone of the UTM projection system;  
 20 an 'x' attribute for indicating an x-coordinate of a first coordinate system;  
 a 'y' attribute for indicating a y-coordinate of the first coordinate system;  
 a 'u' attribute for indicating an x-coordinate of a second coordinate system;  
 a 'v' attribute for indicating a y-coordinate of the second coordinate system; and  
 an 'inputID' attribute for specifying an identification of an element that will feed  
 25 its coordinates into the <mapProj> element.

20. The system as claimed in claim 6, wherein the viewer behavior elements comprise one or more of:

- a <dsvg:zoom> element for scaling a document by a factor;
- a <dsvg:panJ> element for translating a document by an amount; and
- 5 a <dsvg:playSound> element for playing an audio file.

21. The system as claimed in claim 20, wherein the <dsvg:zoom> viewer behavior element comprises:

- an 'id' attribute for assigning a unique name to an element;
- 10 an event attribute for specifying an event that will trigger this action;
- a 'scale' attribute for specifying a scale factor to zoom in or out by;
- an 'absolute' attribute for specifying whether to set the document's scale factor to 'scale' or to its current scale factor multiplied by 'scale';
- a 'cx' attribute for specifying an x-coordinate of a location in the document that
- 15 will stay preserved after the zoom, with respect to a browser window;
- a 'cy' attribute for specifying a y-coordinate of a location in the document that will stay preserved after the zoom, with respect to the browser window; and
- a 'target' attribute for specifying a target document.

20 22. The system as claimed in claim 20, wherein the <dsvg:pan> viewer behavior element comprises:

- an 'id' attribute for assigning a unique name to an element;
- an 'event' attribute for specifying an event that will trigger this action;
- an 'x' attribute for specifying an amount to translate horizontally;
- 25 a 'y' attribute for specifying an amount to translate vertically; and

an 'absolute' attribute for specifying whether to set the document's translation to ('cx', 'cy') or to its current translation plus ('cx', 'cy').

23. The system as claimed in claim 20, wherein the <dsvg:playSound> viewer behavior  
5 element comprises:

an 'id' attribute for assigning a unique name to an element;

an 'event' attribute for specifying an event that will trigger this action; and

an 'xlink:href' attribute for specifying the audio file to play.

10 24. The system as claimed in claim 6, wherein the <dsvg:selection> element comprises:

an 'id' attribute for assigning a unique name to an element;

an 'xlink:href' attribute for referencing a parent element of a skin; and

a 'multiSelect' attribute for specifying whether to allow multiple elements to be  
selected.

15

25. The system as claimed in claim 6, wherein the <dsvg:constraint> element comprises:

an 'id' attribute for assigning a unique name to an element;

a 'target' attribute for specifying a target element whose attribute or style property  
is to be constrained;

20 an 'attributeName' attribute for specifying a name of an attribute to be  
constrained;

a 'propertyName' attribute for specifying a name of a style property to be  
constrained;

a 'value' attribute for specifying a value that the attribute, defined by  
25 'attributeName', or the style property, defined by 'propertyName', is to be given;

a 'scaleImmunity' attribute for specifying immune that the attribute or style property of a the target element is immune to scaling;

a 'zoomImmunity' for specifying that the attribute or style property of the target element is immune to zooming;

5 a 'preserveAspectRatio' attribute for specifying which dimension is to be preserved, thus altering the other dimension so as to preserve the original aspect ratio, which was altered;

an 'hAlign' attribute for specifying a part of the target element, along the x-axis, that is to have its position preserved after executing the constraint;

10 a 'vAlign' attribute for specifying a part of the target element, along the y-axis, that is to have its position preserved after executing the constraint;

a 'width' attribute for specifying a width of the target element;

a 'height' attribute for specifying a height of the target element;

a 'left' attribute for specifying an x-coordinate of a left edge of the target element;

15 a 'right' attribute for specifying an x-coordinate of a right edge of the target element;

a 'top' attribute for specifying a y-coordinate of a top edge of the target element;  
and

a 'bottom' attribute for specifying a y-coordinate of the bottom edge of the target  
20 element.

26. The system as claimed in claim 1, wherein the initialization file contains instructions for traversing each node in the document object model and for searching and calling functions associated with designated elements having names following a predetermined  
25 naming convention.



27. A method of controlling statement flow of a web application, the method comprising the steps of:

searching for a flow control element in a document object model of the web application;

- 5       generating a function name associated with the flow control element;  
calling the generated function name; and  
processing child elements of the flow control element.

28. A method of coordinate mapping of a web application, the method comprising the steps of:

10       searching for a coordinate mapping element in a document object model of the web application;

generating a function name associated with the coordinate mapping element; and  
calling the generated function name.

15

29. A method of manipulating viewer behavior with respect to a web application, the method comprising the steps of:

searching for a viewer behavior element in a document object model of the web application;

- 20       generating a function name associated with the viewer behavior element; and  
calling the generated function name.

30. A method of selecting a group of element in a web application, the method comprising the steps of:

- 25       searching for a selection element in a document object model of the web application;

generating a function name associated with the selection element; and  
calling the generated function name.

31. A method of constraining manipulable attributes of an element in a web application,  
5 the method comprising the steps of:

searching for a constraint element in a document object model of the web  
application;

generating a function name associated with the constraint element; and  
calling the generated function name.

10

32. A method of applying passive behavior to an element of a web application, the  
method comprising the steps of:

searching for a designated attribute of the element in a document object model of  
the web application; and

15

generating a function name associated with the designated attribute; and  
calling the generated function name.

33. A method of manipulating a document object model, the method comprising the steps  
of:

20

searching for a designated control element in the document object model; and  
calling a function associated with the designated control element.

34. The method as claimed in claim 33, wherein the step of searching includes the steps  
of:

25

traversing each node in the document object model; and

determining whether an element has a name which follows a designated naming convention.

35. The method as claimed in claim 33, wherein the designated naming convention  
5 comprises appending a prefix to the name of the designated element.

36. The method as claimed in claim 33, wherein the step of calling a script includes the steps of:

dynamically generating a function name associated with the designated element;  
10 passing an object associated with the designated element as a parameter of the generated function;  
retrieving the attributes of the object; and  
performing a function stored in memory having the generated function name.

15 37. The method as claimed in claim 36, wherein the step of dynamically generating includes the steps of:

determining if the name of the designated element contains a designated prefix;  
generating a function name comprising of the name of the designated element;  
assigning an object associated with the designated element as the parameter of the  
20 function; and  
assigning predetermined instructions of the designated element as steps for the function to perform.

38. The method as claimed in claim 33, wherein the step of calling a script includes the  
25 steps of:

determining which script in a collection of scripts is associated with the designated element; and

calling the script.

39. The method as claimed in claim 33, further comprising the steps of:

searching for a designated attribute in an element in a document object model; and

5 calling a script associated with the designated attribute.

40. The method as claimed in claim 39, wherein the step of searching for a designated attribute comprises the steps of:

searching attributes of an element in a document object model;

10 determining whether an element attribute has a name which follows a designated naming convention.

41. The method as claimed in claim 39, wherein the naming convention comprises appending a prefix to the name of the designated attribute.

15

42. The method as claimed in claim 39, wherein the step of calling a script includes the steps of:

determining if the name of the designated attribute contains a designated prefix;

generating a function name comprising of the name of the designated attribute;

20 assigning an object associated with the designated attribute as the parameter of the function name ; and

assigning predetermined instructions of the designated attribute as steps for a function having the function name to perform.

25 43. The method as claimed in claim 39, wherein the step of calling a script includes the steps of:

dynamically generating a function name associated with the designated attribute;  
passing an object associated with the designated attribute as a parameter of the  
generated function name;  
receiving the attributes of the object; and  
5 performing a function stored in memory having the generated function name.

44. The method as claimed in claim 43, wherein the step of dynamically generating  
comprises the steps of:

determining if the name of the designated attribute contains a designated prefix;  
10 generating a function name comprising of the name of the designated attribute;  
assigning an object associated with the designated attribute as the parameter of the  
function; and  
assigning predetermined instructions of the designated attribute as steps for the  
function to perform.

15

45. The method as claimed in claim 43, wherein the step of calling a script includes the  
steps of:

determining which script in a collection of scripts is associated with the  
designated attribute; and  
20 calling the script.

46. A method of controlling user interface features of a web application, the method  
comprising the steps of:

adding a behavior element as a child of a designated element;  
25 receiving an event which is equal to an event attribute setting in the behavior  
element; and

calling a script associated with the behavior element.

47. Computer readable media storing the instructions and/or statements for use in the execution in a computer of a method of manipulating a document object model of a web application, the method comprising steps of:

searching for a designated element in a document object model; and  
calling a script associated with the designated element.

48. Electronic signals for use in the execution in a computer of a method of manipulating a document object model of a web application, the method comprising steps of:

searching for a designated element in a document object model; and  
calling a script associated with the designated element.

49. A computer program product for use in the execution in a computer of a method for manipulating a document object model of a web application, the computer program product comprising:

a collection of functions for performing actions associated with designated elements; and

an initialization function for directing the processing of one or more designated elements in a document object model.

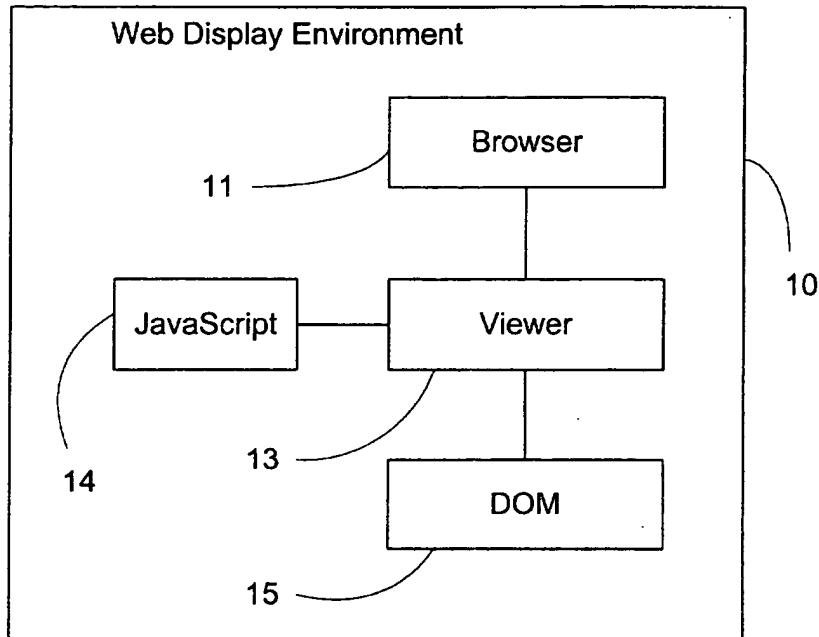


Figure 1  
PRIOR ART

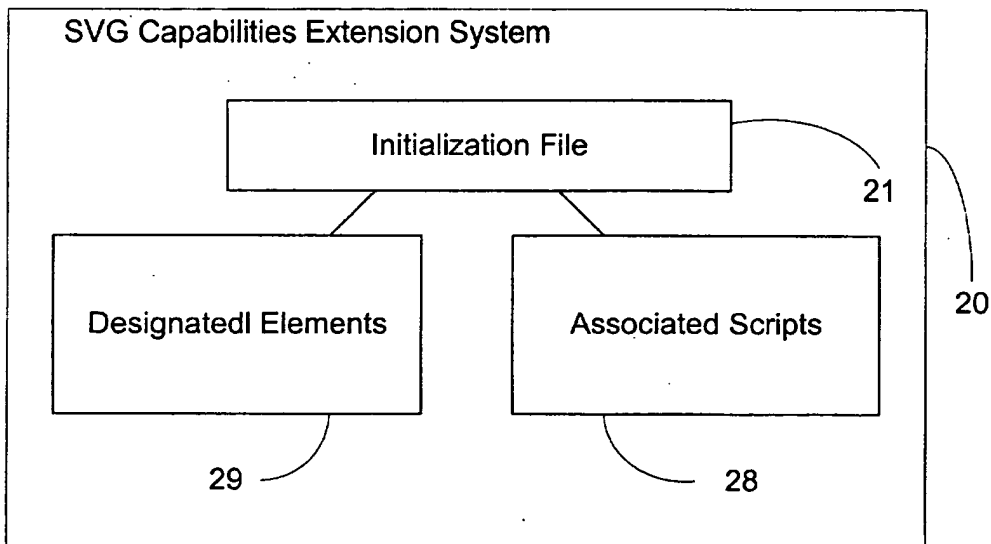


Figure 2

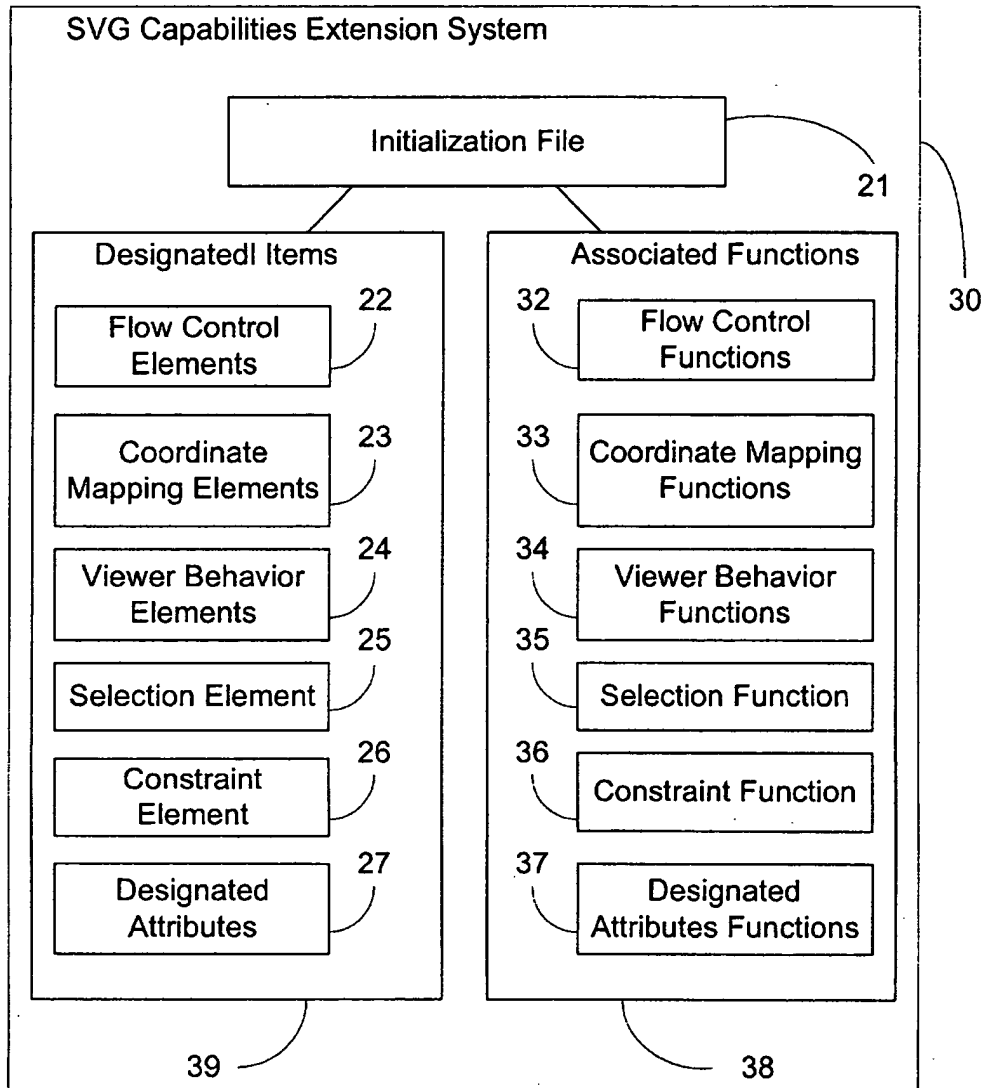


Figure 3



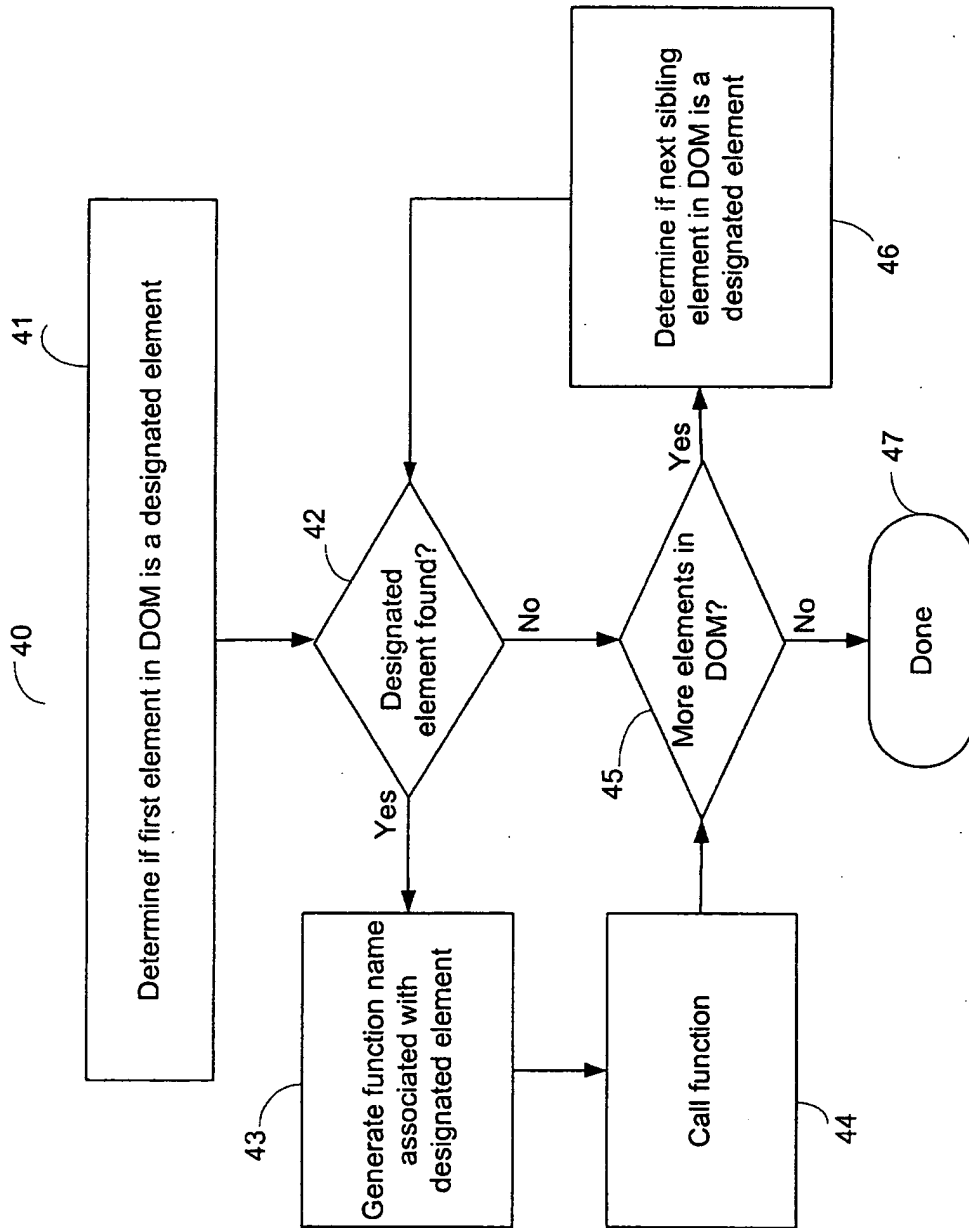


Figure 4

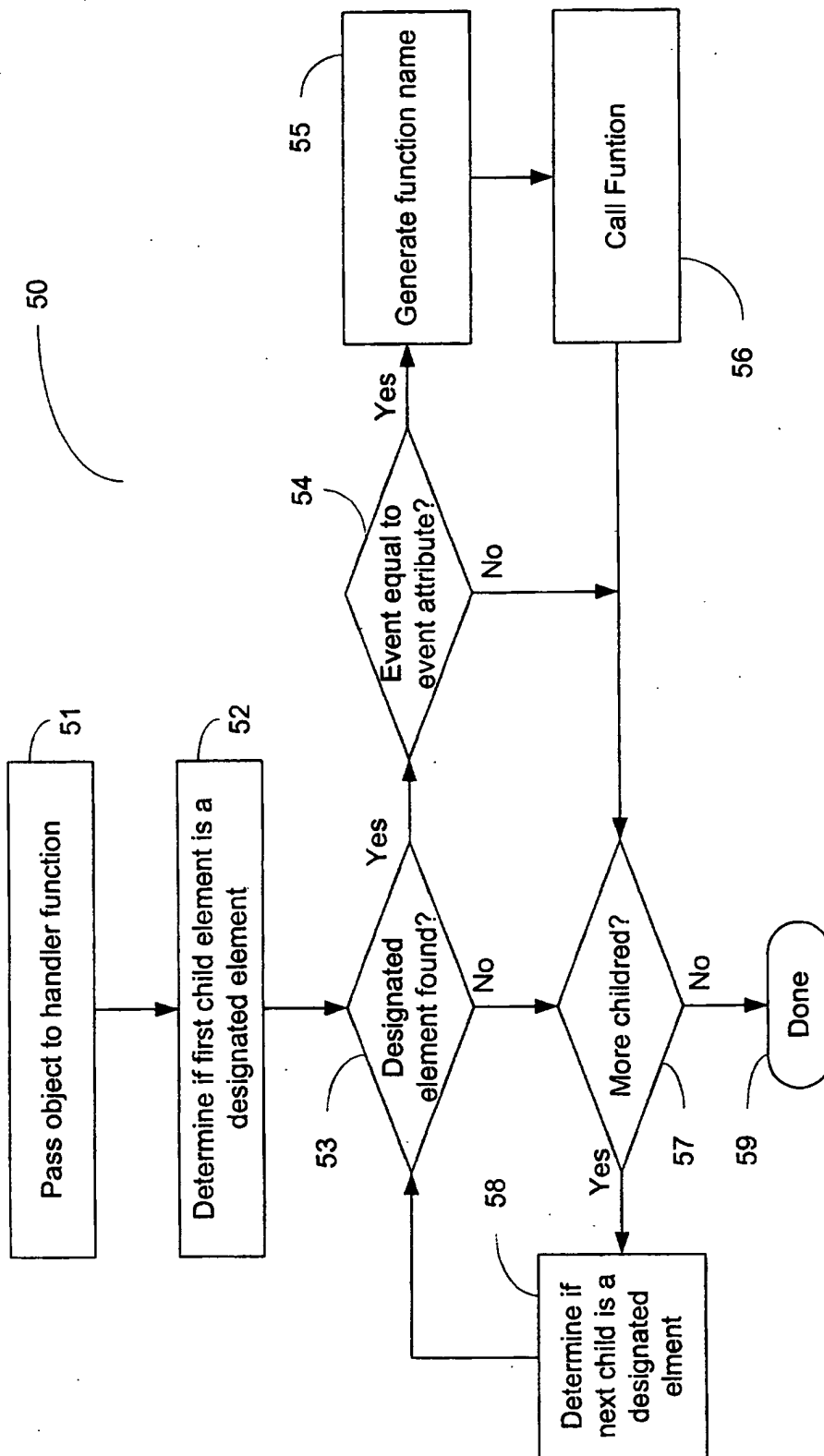


Figure 5

